Simultaneous Feature Aggregating and Hashing for Compact Binary Code Learning

Thanh-Toan Do, Khoa Le, Tuan Hoang, Huu Le, Tam V. Nguyen, Ngai-Man Cheung

***Abstract*—Representing images by compact hash codes is an attractive approach for large-scale content-based image retrieval. In most state-of-the-art hashing-based image retrieval systems, for each image, local descriptors are first aggregated as a global representation vector. This global vector is then subjected to a hashing function to generate a binary hash code. In previous works, the aggregating and the hashing processes are designed independently. Hence these frameworks may generate suboptimal hash codes. In this paper, we first propose a novel unsupervised hashing framework in which feature aggregating and hashing are designed simultaneously and optimized jointly. Specifically, our joint optimization generates aggregated representations that can be better reconstructed by some binary codes. This leads to more discriminative binary hash codes and improved retrieval accuracy. In addition, the proposed method is flexible. It can be extended for supervised hashing. When the data label is available, the framework can be adapted to learn binary codes which minimize the reconstruction loss w.r.t. label vectors. Furthermore, we also propose a fast version of the state-of-the-art hashing method Binary Autoencoder to be used in our proposed frameworks. Extensive experiments on benchmark datasets under various settings show that the proposed methods outperform state-of-the-art unsupervised and supervised hashing methods.**

***Index Terms*—Image search, binary hashing, aggregating, embedding.**

I. Introduction

Content-based image retrieval is an important problem in computer vision with many applications, including visual search [1]–[6], place recognition [7]–[9], camera pose estimation [10]–[12]. State-of-the-art image search systems [1], [2], [13]–[15] include three main steps in computing the image representation: local feature extraction, embedding, and aggregating. The local feature extraction step extracts a set of local features, e.g. SIFT [16], representing the image. The embedding step improves the discriminativeness of the local features by mapping these features into a high-dimensional space [1], [13], [15], [17]. The aggregating (pooling) step converts the set of mapped high dimensional vectors into a single vector representation which usually has the dimensionality of several thousands [1], [13], [15], [17]. In particular, the aggregating step is very important. First, the aggregating step reduces the storage requirement which is one of main concerns in large-scale image search. Second, the aggregated representation vectors enables direct comparison using standard metrics such as Euclidean distance.

Although the aggregated representation reduces the storage and allows simple distance-based comparison, it is not efficient enough for large-scale database which requires very compact representation and fast searching. An attractive approach for achieving these requirements is binary hashing. Specifically, binary hashing encodes image representations into compact binary hash codes, in which distances among data points can be efficiently calculated using bit operations, i.e., XOR and POPCOUNT. Furthermore, binary representations reduce storage significantly.

Although both aggregating and hashing play important roles in large scale image search systems, in most works, the aggregating and hashing steps are designed independently and separately [18]–[20]: First, some aggregation is applied on the local (embedded) features, resulting in a single aggregated representation for each image. Then, the set of aggregated representations is used for learning a hash function which encodes the aggregated representations into compact binary codes. For example, Generalized Max Pooling [21] seeks a representation that can achieve some desirable aggregation property, i.e., equalizing the similarity between the representation and individual local features. This aggregation process does not take into account any aspect of the subsequent hashing, and the resulted representations may not be optimal for hashing, e.g., in the context of unsupervised hashing, the aggregated representation may be difficult to be reconstructed by binary codes. In this work, we propose a novel framework where feature aggregating and hashing are designed simultaneously and optimized jointly. Specifically, in our proposed framework, we aim to compute aggregated representations that not only can achieve some desired aggregation property (equalized similarity) but also can be better reconstructed by some binary codes (in the unsupervised setting) or can better preserve the semantic similarity (in the supervised setting). As the aggregation is more reconstructible (for the unsupervised setting) and can preserve more semantic information (for the supervised setting), the binary codes are discriminative, resulting in improved retrieval performance.

Our specific contributions are: *(i)* In order to accelerate simultaneous learning of aggregating and hashing, we first propose a relaxed version of the state-of-the-art unsupervised hashing Binary Autoencoder [22] to be used in our framework. Instead of solving a NP-hard problem with the hard binary constraint on the outputs of the encoder, we propose to

Thanh-Toan Do is with the University of Liverpool, United Kingdom. E-mail: thanh-toan.do@liverpool.ac.uk

Khoa Le, Tuan Hoang, and Ngai-Man Cheung are with the Singapore University of Technology and Design, Singapore. E-mail: {letandang_khoa,ngaiman_cheung}@sutd.edu.sg, nguyenanhtuan_hoang@mymail.sutd.edu.sg

Huu Le is with Chalmers University of Technology, Sweden. E-mail: huulem@outlook.com

Tam V. Nguyen is with the University of Dayton, United States. E-mail: tamnguyen@udayton.edu

solve the problem with relaxation of the binary constraint, i.e., minimizing the binary quantization loss. In order to minimize this loss, we propose to solve the problem with alternating optimization. This proposed hashing method is not only faster in training but also competitive in retrieval accuracy in comparison to Binary Autoencoder [22]. *(ii)* Our second contribution is a simultaneous feature aggregating and hashing learning approach which takes a set of local (embedded) features[1] as the input and learns the aggregation and the hash function simultaneously. We propose alternating optimization for learning the aggregated features and the hash function. The proposed relaxed Binary Autoencoder is used for learning the hash parameters in the alternating optimization. *(iii)* we extend the framework to supervised hashing by leveraging the label information such that the binary codes preserve the semantic similarity of samples. Furthermore, in the supervised setting, the binary codes of a new testing image can not be directly computed from learned model parameters because the aggregating process requires image label which is not available for the testing image. To overcome this challenge, we propose a novel simple yet powerful solution that learns a mapping from original aggregated features to learned aggregated features. The learned mapping is then used in the process of computing binary codes for new images. *(iv)* We perform solid experiments on different image retrieval benchmark datasets to evaluate the proposed frameworks. We also evaluate the proposed methods with different state-of-the-art image features under different configurations. The experimental results show that the proposed simultaneous learning outperforms other recent unsupervised and supervised hashing methods.

A preliminary version of this work has been presented in [23]. The extensions in this current version are: Firstly, the introduction and the related work sections are fully revised to clearly describe our contributions and to thoroughly cover recent works. Secondly, we provide extensive comparisons between the proposed unsupervised framework to recent state-of-the-art deep learning-based unsupervised hashing methods. Thirdly, we adapt the proposed unsupervised framework to supervised hashing. In particular, we reformulate problem formulations and optimization processes by using the label information to supervise the learning. The binary codes are learned such that they not only encourage the aggregating property but also optimize for a linear classifier. We also propose a novel simple yet powerful solution to compute the binary codes of testing images. In addition, we also provide the asymptotic complexity the proposed method. Finally, we extensively evaluate and compare the proposed supervised framework to both state-of-the-art non-deep-based and deep-based supervised hashing methods under various configurations, i.e., traditional setting and unseen class setting. The experimental results show that our supervised method outperforms the state-of-the-art supervised hashing methods.

The remaining of this paper is organized as follows. Section II presents related works. Section III introduces the Relaxed Binary Autoencoder. Sections IV and V introduce and evaluate the proposed simultaneous feature aggregating and hashing (SAH) for unsupervised hashing, respectively. Sections VI and VII introduce and evaluate the proposed simultaneous feature aggregating and supervised hashing (SASH), respectively. Section VIII concludes the paper.

[1]In this work, the embedding is applied when SIFT features are used.

TABLE I
NOTATIONS AND THEIR CORRESPONDING MEANINGS.

Notation	Meaning
$\mathbf{X}$	$\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{m} \in \mathbb{R}^{D\times m}$: set of m training samples; each column of $\mathbf{X}$ corresponds to one sample
$\mathbf{Y}$	$\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^{m} \in \mathbb{R}^{C\times m}$: set of label vectors for supervised setting
$\mathbf{B}$	$\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^{m} \in \{-1,+1\}^{L\times m}$: binary code matrix
L	Number of bits to encode a sample
$\mathbf{W}_1, \mathbf{c}_1$	$\mathbf{W}_1 \in \mathbb{R}^{L\times D}, \mathbf{c}_1 \in \mathbb{R}^{L\times 1}$: weight and bias of encoder
$\mathbf{W}_2, \mathbf{c}_2$	weight and bias of decoder. • $\mathbf{W}_2 \in \mathbb{R}^{D\times L}, \mathbf{c}_2 \in \mathbb{R}^{D\times 1}$: unsupervised setting • $\mathbf{W}_2 \in \mathbb{R}^{C\times L}, \mathbf{c}_2 \in \mathbb{R}^{C\times 1}$: supervised setting
$\mathcal{V}$	$\mathcal{V} = \{\mathbf{V}_i\}_{i=1}^{m}$; $\mathbf{V}_i \in \mathbb{R}^{D\times n_i}$ is set of local (embedded) representations of image i; n_i is number of local descriptors of image i
Φ	$\Phi = \{\varphi_i\}_{i=1}^{m} \in \mathbb{R}^{D\times m}$: set of m aggregated vectors; φ_i corresponds to aggregated vector of image i
$\mathbf{1}$	column vector with all $1s$ elements
$\mathbf{I}$	identity matrix

II. Related work

In order to make the paper clear and easy to follow, we summarize the used notations in Table I. Two main components of the proposed simultaneous learning are aggregating and hashing. For aggregating, we rely on the state-of-the-art Generalized Max Pooling [21]. For hashing, we propose a relaxed version of Binary Autoencoder [22]. This section presents a brief overview of Generalized Max Pooling [21] and hashing methods.

A. Generalized Max Pooling (GMP) [21]

Sum-pooling [24] and max-pooling [25], [26] are two common methods for aggregating set of local (embedded) vectors of an image to a single vector. However, sum-pooling lacks discriminability because the aggregated vector is more influenced by frequently-occurring uninformative descriptors than rarely-occurring informative ones. Max-pooling equalizes the influence of frequent and rare descriptors. However, classical max-pooling approaches can only be applied to BoW or sparse coding features. To overcome this challenge, in [1] and [21] the authors concurrently introduced a generalization of max-pooling (i.e., Generalized Max Pooling (GMP) [21])[2] that can be applied to general features such as VLAD [27], Temb [1], Fisher vector [28]. The main idea of GMP is to equalize the similarity between each local embedded vector and the aggregated representation. In [1], [15], the authors showed that GMP achieves better retrieval accuracy than sum-pooling.

Given $\mathbf{V} \in \mathbb{R}^{D\times n}$, the set of n embedded vectors of an image (each embedded vector has dimensionality D), GMP finds the aggregated representation φ which equalizes the

[2]In [1], the authors named their method as *democratic aggregation*. It actually shares similar idea to *generalized max pooling* [21]

similarity (i.e. the dot-product) between each column of $\mathbf{V}$ and φ by solving the following optimization

$$\min_{\varphi} \left(\left\| \mathbf{V}^T \varphi - \mathbf{1} \right\|^2 + \mu \left\| \varphi \right\|^2 \right) \quad (1)$$

(1) is a ridge regression problem which solution is

$$\varphi = \left(\mathbf{V}\mathbf{V}^T + \mu \mathbf{I} \right)^{-1} \mathbf{V}\mathbf{1} \quad (2)$$

B. Hashing methods

Existing binary hashing methods can be categorized as data-independent and data-dependent schemes [29]–[31]. Data-independent hashing methods [32]–[35] rely on random projections for constructing hash functions. Although representative data-independent hashing methods such as Locality-Sensitive Hashing (LSH) [32] and its kernelized versions [33], [34] have theoretical guarantees that the more similar data would have higher probability to be mapped into similar binary codes, they require long codes to achieve high precision. Different from data-independent approaches, data-dependent hashing methods use available training data for learning hash functions in unsupervised or supervised manner and they usually achieve better retrieval results than data-independent methods. The unsupervised hashing methods [22], [36]–[40] try to preserve the neighbor similarity of samples in Hamming space without label information. The representative unsupervised hashing methods are Iterative Quantization (ITQ) [37], Spherical Hashing (SPH) [39], K-means Hashing (KMH) [38], etc. The supervised hashing methods [41]–[45] try to preserve the label similarity of samples using labeled training data. The representative supervised hashing methods are Kernel-Based Supervised Hashing (KSH) [42], Semi-supervised Hashing (SSH) [46], Supervised Discrete Hashing (SDH) [44], Asymmetric Inner-product Binary Coding (AIBC) [47], Graph Convolutional Network Hashing (GCNH) [48], etc.

Most of the previous hashing methods are originally designed and experimented on hand-crafted features which may limit their performance in practical applications. Recently, to leverage the power of deep convolutional neural networks (CNNs) [49]–[51], many deep unsupervised and supervised hashing methods have been proposed. In [52] the authors proposed a two-step supervised hashing method which learns a deep CNN based hash function with the pre-computed binary codes. In [53]–[58] the authors proposed end-to-end deep supervised hashing methods in which the image features and the hash codes are simultaneously learned. Most of those models consist of a deep CNN for image feature extraction and a binary quantization component that tries to approximate the *sgn* function. Different from supervised setting, there are few end-to-end hashing which are proposed for unsupervised setting. In [59], the authors proposed an end-to-end deep learning framework for unsupervised hashing. The network is trained to produce hash codes that minimize the quantization loss w.r.t. the output of the last VGG's [51] fully connected layer. Recently, in [60] the authors proposed an unsupervised deep hashing method that alternatingly proceeds over three training modules: deep hash model training, similarity graph updating and binary code optimization. Different from previous deep hashing methods that try to simultaneously learn image features and binary codes, our work uses a pre-trained deep model to extract local image representations. These representations are used as inputs for the simultaneous learning of the aggregated representation and the binary codes.

One of problems which makes the binary hashing difficult is the binary constraint on the codes, i.e., the outputs of the hash functions have to be binary. Generally, this binary constraint leads to an NP-hard mixed-integer optimization problem. In order to overcome this difficulty, several approaches have been proposed in the literature. In Deep Hashing (DH) [40], the binary constraint is handled by applying the $sign$ function on the outputs of the last layer of the network. The authors assumed that the $sign$ function is differentiable everywhere during training. In Iterative Quantization [37], the binary constraint is relaxed by minimizing the binary quantization loss. In Binary Autoencoder [22] the binary constraint is handled by using the $sign$ function. In order to overcome the non-differentiable of the $sign$ function, the authors [22] use binary SVMs to learn the model parameters. In [61], the authors also used the $sign$ function to handle the binary constraint. To deal with the non-differentiable problem of the sign function, the authors proposed to use the hinge loss to approximate the sign function. In [62], to handle the binary constraint, the authors relied on the idea of minimizing the binary quantization error which is achieved by an alternating optimization over the real-valued network weights and auxiliary binary variables.

There is another research topic that is related to binary hashing, i.e., training neural networks with binary weights. In this problem, the network weights are constrained to be binary. In the recent work, Binary Connect [63], the authors use a stochastic binarization function to binarize the network weights. To handle the gradient problem of the binarization function which is zero almost everywhere, the authors utilized the straight-through estimator [64] to approximate the gradient of the function. It is worth noting that the activation outputs of the Binary Connect [63] are still real-valued, which is different from hashing problem which aims to produce binary outputs. In the following, we brief the two most related hashing methods to our work, i.e., Iterative Quantization (ITQ) [37] and Binary Autoencoder (BA) [22].

Iterative Quantization (ITQ) [37]: In [37], the authors propose ITQ which is a two-step hashing method. In the first step, ITQ computes continuous low-dimensional codes by applying PCA to the data. In the second step, it finds a rotation that makes the PCA codes as close as possible to binary values. The second step is actually an Orthogonal Procrustes problem [65] in which one tries to find a rotation to align one point set with another and it has a closed-form solution by using SVD. ITQ is a postprocessing of the PCA codes and it can be seen as a suboptimal approach to optimizing a binary autoencoder, where the binary constraints are relaxed during the optimization (i.e., in the PCA step), and then one projects the continuous codes back to the binary space.

Binary Autoencoder (BA) [22]: In [22], instead of ignoring binary constraints during the dimensionality reduction and then binarizing the continuous codes, the authors propose a

joint optimization. Specifically, in order to compute the binary codes, the authors minimize the following optimization

$$\min_{\mathbf{h},\mathbf{f},\mathbf{Z}} \sum_{i=1}^{m} \left(\|\mathbf{x}_i - \mathbf{f}(\mathbf{z}_i)\|^2 + \mu \|\mathbf{z}_i - \mathbf{h}(\mathbf{x}_i)\|^2 \right) \quad (3)$$

$$\text{s.t. } \mathbf{z}_i \in \{-1, 1\}^L, i = 1, ..., m \quad (4)$$

where $\mathbf{h} = sgn(\mathbf{W}_1\mathbf{x} + \mathbf{c}_1)$ and $\mathbf{f}$ are encoder and decoder, respectively. By having sgn, the encoder will output binary codes. In the training of BA, the authors compute each variable $\mathbf{f}, \mathbf{h}, \mathbf{Z}$ at a time while holding the other fixed. The authors show that the BA outperforms state-of-the-art unsupervised hashing methods. However, the disadvantage of BA is the time-consuming training which is mainly caused by the computing of $\mathbf{h}$ and $\mathbf{Z}$. As $\mathbf{h}$ involves sgn, it cannot be solved analytically. Hence, when computing $\mathbf{h}$, the authors cast the problem as the learning of L separated linear SVM classifiers, i.e., for each $l = 1, ..., L$, they fit a linear SVM to $(\mathbf{X}, \mathbf{Z}_{l,.})$. When computing $\mathbf{Z}$, the authors solve for each sample $\mathbf{x}_i$ independently. Solving $\mathbf{z}_i$ in (3) for each sample under the binary constraint (4) is NP-hard. To handle this, the authors first solve the problem with the relaxed constraint $\mathbf{z}_i \in [-1, 1]$, resulting a continuous solution. They then apply the following procedure several times for getting $\mathbf{z}_i$: for each bit from 1 to L, they evaluate the objective function with the bit equals to -1 or 1 with all remaining elements fixed and pick the best value for that bit. The asymptotic complexity for computing $\mathbf{Z}$ over all samples is $\mathcal{O}(mL^3)$.

In the following, we introduce our efficient Relaxed Binary Autoencoder algorithm (Section III) which will be used in the proposed simultaneous feature aggregating and hashing framework (Section IV).

III. Relaxed Binary Autoencoder (RBA)

A. Formulation

In order to achieve binary codes, we propose to solve the following constrained optimization

$$\min_{\{\mathbf{W}_i, \mathbf{c}_i\}_{i=1}^2} J = \frac{1}{2} \left\| \mathbf{X} - \left(\mathbf{W}_2(\mathbf{W}_1\mathbf{X} + \mathbf{c}_1\mathbf{1}^T) + \mathbf{c}_2\mathbf{1}^T \right) \right\|^2 + \frac{\beta}{2} \left(\|\mathbf{W}_1\|^2 + \|\mathbf{W}_2\|^2 \right) \quad (5)$$

$$\text{s.t. } \mathbf{W}_1\mathbf{X} + \mathbf{c}_1\mathbf{1}^T \in \{-1, 1\}^{L \times m} \quad (6)$$

The constraint (6) makes sure the output of the encoder is binary. The first term of (5) makes sure the binary codes provide a good reconstruction of the input, so it encourages (dis)similar inputs map to (dis)similar binary codes. The second term is a regularization that tends to decrease the magnitude of the weights, so it helps to prevent overfitting.

Solving (5) under (6) is difficult due to the binary constraint. In order to overcome this challenge, we propose to solve the relaxed version of the binary constraint, i.e., minimizing the binary quantization loss of the encoder. The proposed method is named as *Relaxed Binary Autoencoder* (RBA). Specifically, inspired from the quadratic penalty method for constrained optimization [66], we introduce a new auxiliary variable $\mathbf{B}$ and solve the following the optimization

$$\min_{\{\mathbf{W}_i, \mathbf{c}_i\}_{i=1}^2, \mathbf{B}} J = \frac{1}{2} \left\| \mathbf{X} - \left(\mathbf{W}_2\mathbf{B} + \mathbf{c}_2\mathbf{1}^T \right) \right\|^2 + \frac{\lambda}{2} \left\| \mathbf{B} - (\mathbf{W}_1\mathbf{X} + \mathbf{c}_1\mathbf{1}^T) \right\|^2 + \frac{\beta}{2} \left(\|\mathbf{W}_1\|^2 + \|\mathbf{W}_2\|^2 \right) \quad (7)$$

$$\text{s.t. } \mathbf{B} \in \{-1, 1\}^{L \times m} \quad (8)$$

The benefit of the auxiliary variable $\mathbf{B}$ is that we can decompose the difficult constrained optimization problem (5) into simpler sub-problems. We use alternating optimization on these sub-problems as will be discussed in detail.

An important difference between the proposed RBA and the original BA is that our encoder does not involve sgn function. The second term of (7) forces the output of encoder close to binary values, i.e., it minimizes the binary quantization loss, while the first term still ensures good reconstruction loss. By setting the penalty parameter λ sufficiently large, we penalize the binary constraint violation severely, thereby forcing the solution of (7) closer to the feasible region of the original problem (5).

B. Optimization

In order to solve for $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2, \mathbf{B}$ in (7) under constraint (8), we solve each variable at a time while holding the other fixed.

$(\mathbf{W}, \mathbf{c})$-step: When fixing $\mathbf{c}_1, \mathbf{c}_2$ and $\mathbf{B}$, we have the closed forms for $\mathbf{W}_1, \mathbf{W}_2$ as follows

$$\mathbf{W}_1 = \lambda \left(\mathbf{B} - \mathbf{c}_1\mathbf{1}^T \right) \mathbf{X}^T \left(\lambda\mathbf{X}\mathbf{X}^T + \beta\mathbf{I} \right)^{-1} \quad (9)$$

$$\mathbf{W}_2 = \left(\mathbf{X} - \mathbf{c}_2\mathbf{1}^T \right) \mathbf{B}^T \left(\mathbf{B}\mathbf{B}^T + \beta\mathbf{I} \right)^{-1} \quad (10)$$

When fixing $\mathbf{W}_1, \mathbf{W}_2$ and $\mathbf{B}$, we have the closed forms for $\mathbf{c}_1, \mathbf{c}_2$ as follows

$$\mathbf{c}_1 = \frac{1}{m} \left(\mathbf{B} - \mathbf{W}_1\mathbf{X} \right) \mathbf{1} \quad (11)$$

$$\mathbf{c}_2 = \frac{1}{m} \left(\mathbf{X} - \mathbf{W}_2\mathbf{B} \right) \mathbf{1} \quad (12)$$

Note that in (9), the term $\mathbf{X}^T \left(\lambda\mathbf{X}\mathbf{X}^T + \beta\mathbf{I} \right)^{-1}$ is a constant matrix and it is computed only one time.

$\mathbf{B}$-step: When fixing the weight and the bias, by defining $\widetilde{\mathbf{X}}$ and $\mathbf{H}$ as follows

$$\widetilde{\mathbf{X}} = \mathbf{X} - \mathbf{c}_2\mathbf{1}^T \quad (13)$$

$$\mathbf{H} = \mathbf{W}_1\mathbf{X} + \mathbf{c}_1\mathbf{1}^T \quad (14)$$

we can rewrite (7) as

$$\left\| \widetilde{\mathbf{X}} - \mathbf{W}_2\mathbf{B} \right\|^2 + \lambda \|\mathbf{H} - \mathbf{B}\|^2 \quad (15)$$

$$\text{s.t. } \mathbf{B} \in \{-1, 1\}^{L \times m} \quad (16)$$

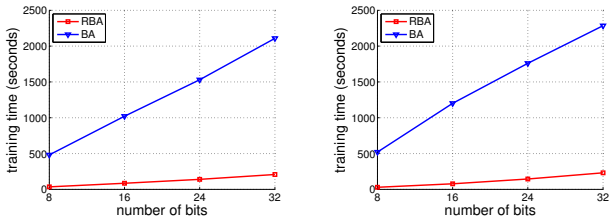
Inspired by the recent progress of discrete optimization [44], we use coordinate descent approach for solving $\mathbf{B}$, i.e., we solve one row of $\mathbf{B}$ each time while fixing all other rows. Specifically, let $\mathbf{Q} = \mathbf{W}_2^T\widetilde{\mathbf{X}} + \lambda\mathbf{H}$; for $k = 1, ..., L$, let $\mathbf{w}_k$ be k^{th} column of $\mathbf{W}_2$; $\overline{\mathbf{W}}_2$ be matrix $\mathbf{W}_2$ excluding $\mathbf{w}_k$; $\mathbf{q}_k$

Algorithm 1 Relaxed Binary Autoencoder (RBA)

Input:
$\mathbf{X}$: training data; L: code length; T_1: maximum iteration number; parameters λ, β

Output:
Parameters $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2$

Initialize $\mathbf{B}^{(0)} \in \{1, 1\}^{L \times m}$ using ITQ [37]
Initialize $\mathbf{c}_1^{(0)} = \mathbf{0}$, $\mathbf{c}_2^{(0)} = \mathbf{0}$
for $t = 1 \rightarrow T_1$ **do**
    Fix $\mathbf{B}^{(t-1)}, \mathbf{c}_1^{(t-1)}, \mathbf{c}_2^{(t-1)}$, solve $\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}$
    Fix $\mathbf{B}^{(t-1)}, \mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}$, solve $\mathbf{c}_1^{(t)}, \mathbf{c}_2^{(t)}$
    Fix $\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{c}_2^{(t)}$, solve $\mathbf{B}^{(t)}$ by **B-step**
end for
Return $\mathbf{W}_1^{(T_1)}, \mathbf{W}_2^{(T_1)}, \mathbf{c}_1^{(T_1)}, \mathbf{c}_2^{(T_1)}$



(a) CIFAR10 (b) SIFT1M

Fig. 1. Training time of BA and RBA on CIFAR10 and SIFT1M

be k^{th} column of $\mathbf{Q}^T$; $\mathbf{b}_k^T$ be k^{th} row of $\mathbf{B}$; $\overline{\mathbf{B}}$ be matrix $\mathbf{B}$ excluding $\mathbf{b}_k^T$. We have the closed-form solution for $\mathbf{b}_k^T$ as

$$\mathbf{b}_k^T = sgn\left(\mathbf{q}_k^T - \mathbf{w}_k^T \overline{\mathbf{W}}_2 \overline{\mathbf{B}}\right) \quad (17)$$

The proposed RBA is summarized in Algorithm 1. In the Algorithm 1, $\mathbf{B}^{(t)}$, $\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)}$ are values at t^{th} iteration. After learning $(\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2)$, given a new vector $\mathbf{x}$, we pass $\mathbf{x}$ to the encoder, i.e., $h = \mathbf{W}_1\mathbf{x} + \mathbf{c}_1$, and round the values of h to $\{-1, 1\}$, resulting binary codes.

Comparison to Binary Autoencoder (BA) [22]: There are two main advances of the proposed RBA (7) over BA (3). First, our encoder does not involve the sgn function. Hence, during the iterative optimization, instead of using SVM for learning the encoder as in BA, we have an analytic solution ((9) and (11)) for the encoder. Second, when solving for $\mathbf{B}$, instead of solving each sample at a time as in BA, we solve all samples at the same time by adapting the recent advance discrete optimization technique [44]. The asymptotic complexity for computing one row of $\mathbf{B}$, i.e. (17), is $\mathcal{O}(mL)$. Hence the asymptotic complexity for computing $\mathbf{B}$ is only $\mathcal{O}(mL^2)$ which is less than $\mathcal{O}(mL^3)$ of BA. These two advances make the training of RBA is faster than BA.

C. Evaluation of Relaxed Binary Autoencoder (RBA)

This section evaluates the proposed RBA and compares it to the following state-of-the-art unsupervised hashing methods: Iterative Quantization (ITQ) [37], Binary Autoencoder (BA) [22], K-means Hashing (KMH) [38], Spherical Hashing (SPH) [39]. For all compared methods, we use the implementations and the suggested parameters provided by the authors. The values of λ, β and the number of iteration T_1 in the Algorithm 1 are empirically set by cross validation as $10^{-2}, 1$ and 10, respectively. The BA [22] and the proposed RBA required an initialization for the binary code. To make a fair comparison, we follow [22], i.e., using ITQ [37] for the initialization.

1) Dataset and evaluation protocol:

Dataset: We conduct experiments on CIFAR10 [67], MNIST [68] and SIFT1M [69] datasets which are widely used in evaluating hashing methods [22], [37].

CIFAR10 dataset [67] consists of 60,000 images of 10 classes. The dataset is split into training and test sets, with $50,000$ and $10,000$ images, respectively. Each image is represented by 320 dimensional GIST feature [70].

MNIST dataset [68] consists of 70,000 handwritten digit images of 10 classes. The dataset is split into training and test sets, with $60,000$ and $10,000$ images, respectively. Each image is represented by a 784 dimensional gray-scale feature vector.

SIFT1M dataset [69] contains 128 dimensional SIFT vectors [16]. There are 1M vectors used as database for retrieval, 100K vectors for training, and 10K vectors for query.

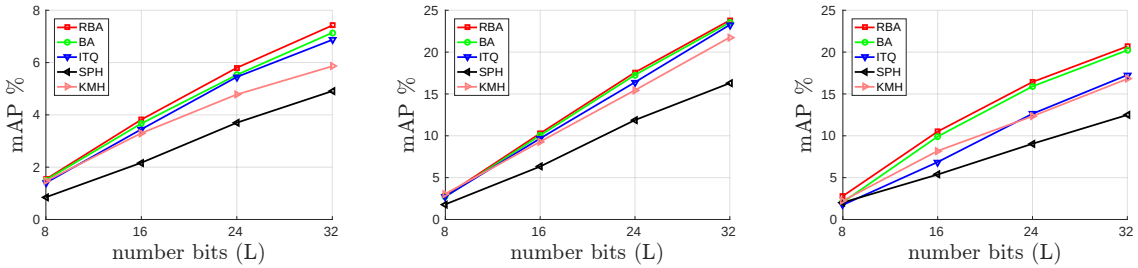
Evaluation protocol: In order to create ground truth for queries, we follow [22], [37] in which the Euclidean nearest neighbors are used. The number of ground truths is set as in [22]. For each query in CIFAR10 and MNIST datasets, its 50 Euclidean nearest neighbors are used as ground truths; for each query in the large scale dataset SIFT1M, its 10,000 Euclidean nearest neighbors are used as ground truths. Follow the state of the art [22], [37], the performance of methods is measured by mAP. Note that as computing mAP is slow on the large scale dataset SIFT1M, we consider top 10,000 returned neighbors when computing mAP.

2) Experimental results:

Training time of RBA and BA: In this experiment, we empirically compare the training time of RBA and BA. The experiments are carried out on a processor core (Xeon E5-2600/2.60GHz). It is worth noting that the implementation of RBA is in Matlab, while BA optimizes the implementation by using mex-files at the encoder learning step. The comparative training time on CIFAR10 and SIFT1M datasets is showed in Figure 1. The results show that RBA is more than ten times faster training than BA for all code lengths on both datasets.

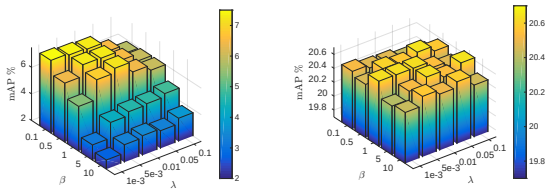
Retrieval results: Figure 2 shows the comparative mAP between methods. We find the following observations are consistent for all three datasets. At all code lengths, the proposed RBA outperforms or is competitive with the state-of-the-art BA. This result confirms the advance of our approach for computing encoder (i.e., closed-form) and **B-step** (i.e. using coordinate descent with closed-form for each row). The results in Figure 2 also confirm the superior performance of BA and RBA over other methods. The improvements are more clear on the large scale SIFT1M dataset.

Effects of hyper-parameters: Figure 3 shows the retrieval performance of RBA with different β and λ values (in Eq. 7) on CIFAR10 and SIFT1M datasets. We can observe that for CIFAR10, RBA generally achieves the best performance when $\beta \in [0.1, 1]$ and $\lambda \in [5 \times 10^{-3}, 5 \times 10^{-2}]$. While for SIFT1M, RBA is pretty robust to $\beta \in [0.1, 10]$ and $\lambda \in [0.001, 0.1]$.



(a) CIFAR10 (b) MNIST (c) SIFT1M

Fig. 2. mAP comparison between RBA and state-of-the-art unsupervised hashing methods on CIFAR10, MNIST, and SIFT1M



(a) CIFAR10 (b) SIFT1M

Fig. 3. The retrieval performance of RBA on CIFAR10 and SIFT1M for 32 bits when varying β and λ.

IV. Simultaneous Feature Aggregating and Hashing (SAH)

A. Formulation

Our goal is to simultaneously learn the aggregated vector representing an image and the hashing function, given the set of local image representations. For simultaneous learning, the learned aggregated vectors and the hash parameters should ensure desired properties of both aggregating and hashing. Specifically, *aggregating property*: (i) for each image i, the dot-product similarity between the aggregated vector φ_i and each local vector of $\mathbf{V}_i$ should be a constant; *hashing properties*: (ii) the outputs of the encoder are binary and (iii) the binary codes should preserve the similarity between image representations. In order to achieve these properties, we formulate the simultaneous learning as the following optimization

$$\min_{\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2,\Phi} \frac{1}{2}\left\|\Phi-\left(\mathbf{W}_2(\mathbf{W}_1\Phi+\mathbf{c}_1\mathbf{1}^T)+\mathbf{c}_2\mathbf{1}^T\right)\right\|^2 + \frac{\beta}{2}\left(\|\mathbf{W}_1\|^2+\|\mathbf{W}_2\|^2\right)+\frac{\gamma}{2}\sum_{i=1}^{m}\left(\left\|\mathbf{V}_i^T\varphi_i-\mathbf{1}\right\|^2+\mu\|\varphi_i\|^2\right) \quad (18)$$

$$\text{s.t. } \mathbf{W}_1\Phi+\mathbf{c}_1\mathbf{1}^T\in\{-1,1\}^{L\times m} \quad (19)$$

The proposed constrained objective function (18) has a clear meaning. The first term of (18) ensures a good reconstruction of Φ, hence it encourages the similarity preserving (the property iii). The binary constraint (19) ensures the binary outputs of encoder (the property ii). Finally, the third term encourages the learned aggregated representation equals the similarities between φ_i and different columns of $\mathbf{V}_i$ by forcing their inner product to be 1 (the property i).

B. Optimization

In order to solve (18) under constraint (19), we propose to iteratively optimize it by alternatingly optimizing w.r.t. hashing parameters $(\mathbf{W},\mathbf{c})$ and aggregated representation Φ while holding the other fixed.

Φ-step: When fixing $\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2$ and solving for Φ, we can solve over each φ_i independently. Specifically, for each sample $i=1,...,m$, we solve the following relaxed problem by skipping the binary constraint

$$\min_{\varphi_i}\frac{1}{2}\left\|\varphi_i-(\mathbf{W}_2(\mathbf{W}_1\varphi_i+\mathbf{c}_1)+\mathbf{c}_2)\right\|^2 + \frac{\gamma}{2}\left(\left\|\mathbf{V}_i^T\varphi_i-\mathbf{1}\right\|^2+\mu\left\|\varphi_i\right\|^2\right) \quad (20)$$

By solving (20), we find φ_i which satisfies the properties (i) and (ii), i.e., φ_i not only ensures the aggregating property but also minimize the reconstruction error w.r.t. the fixed hashing parameters. (20) is actually a l_2 regularized least squares problem, hence we achieve the analytic solution as

$$\varphi_i = \left((\mathbf{I}-\mathbf{W}_2\mathbf{W}_1)^T(\mathbf{I}-\mathbf{W}_2\mathbf{W}_1)+\gamma\mathbf{V}_i\mathbf{V}_i^T+\gamma\mu\mathbf{I}\right)^{-1} \times\left(\gamma\mathbf{V}_i\mathbf{1}+(\mathbf{I}-\mathbf{W}_2\mathbf{W}_1)^T(\mathbf{W}_2\mathbf{c}_1+\mathbf{c}_2)\right) \quad (21)$$

$(\mathbf{W},\mathbf{c})$-step: When fixing Φ and solving for $(\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2)$, (18) under the constraint (19) is equivalent to the following optimization

$$\min_{\{\mathbf{W}_i,\mathbf{c}_i\}_{i=1}^2}\frac{1}{2}\left\|\Phi-\left(\mathbf{W}_2(\mathbf{W}_1\Phi+\mathbf{c}_1\mathbf{1}^T)+\mathbf{c}_2\mathbf{1}^T\right)\right\|^2 + \frac{\beta}{2}\left(\|\mathbf{W}_1\|^2+\|\mathbf{W}_2\|^2\right) \quad (22)$$

$$\text{s.t. } \mathbf{W}_1\Phi+\mathbf{c}_1\mathbf{1}^T\in\{-1,1\}^{L\times m} \quad (23)$$

By solving (22) under the constraint (23), we find hash parameters which satisfy the properties (ii) and (iii), i.e., they not only ensure the binary outputs of the encoder but also minimize the reconstruction error w.r.t. the fixed aggregated representation Φ. (22) and (23) have same forms as (5) and (6). Hence, we solve this optimization with the proposed Relaxed Binary Autoencoder (Section III). Specifically, we use the Algorithm 1 for solving $(\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2)$ in which Φ is used as the training data.

The proposed simultaneous feature aggregating and hashing is presented in the Algorithm 2. In the Algorithm 2, $\Phi^{(t)}$, $\mathbf{W}_1^{(t)},\mathbf{c}_1^{(t)},\mathbf{W}_2^{(t)},\mathbf{c}_2^{(t)}$ are values at t^{th} iteration. After learning $\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2$, given set of local features of a new image,

Algorithm 2 Simultaneous feature Aggregating and Hashing (SAH)

Input:
$\mathcal{V} = \{\mathbf{V}_i\}_{i=1}^m$: training data; L: code length; T, T_1: maximum iteration numbers for SAH and RBA (Algorithm 1), respectively; parameters $\lambda, \beta, \gamma, \mu$.

Output:
Parameters $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2$

Initialize $\Phi^{(0)} = \{\varphi_i\}_{i=1}^m$ with Generalized Max Pooling (2)
for $t = 1 \rightarrow T$ **do**
    Fix $\Phi^{(t-1)}$, solve $(\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)})$ using Algorithm 1 (which uses $\Phi^{(t-1)}$ as inputs for training)
    Fix $(\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)})$, solve $\Phi^{(t)}$ using Φ**-step**.
end for
Return $\mathbf{W}_1^{(T)}, \mathbf{W}_2^{(T)}, \mathbf{c}_1^{(T)}, \mathbf{c}_2^{(T)}$

we first compute its aggregated representation φ using (21). We then pass φ to the encoder to compute the binary code.

V. Evaluation of Simultaneous Feature Aggregating and Hashing (SAH)

This section evaluates and compares the proposed SAH to the following state-of-the-art unsupervised hashing methods: Iterative Quantization (ITQ) [37], Binary Autoencoder (BA) [22] and the proposed RBA, Spherical Hashing (SPH) [39], K-means Hashing (KMH) [38]. For all compared methods, we use the implementations and the suggested parameters provided by the authors. The values of λ, β, γ, and μ are set by cross validation as 10^{-2}, 10^{-1}, 10, and 10^2, respectively.

A. Dataset

We conduct experiments on Holidays [71] and Oxford5k [72] datasets which are widely used in evaluating image retrieval systems [1], [2], [13].

Holidays: The Holidays dataset [71] consists of 1,491 images of different locations and objects, 500 of them being used as queries. Follow standard configuration [1], [2], when evaluating, we remove the query from the ranked list. For the training dataset, we follow [1], [2], i.e., using 10k images from the independent dataset Flickr60k provided with the Holidays.

Holidays+Flickr100k: In order to evaluate the proposed method on large scale, we merge Holidays dataset with 100k images downloaded from Flickr [73], forming the Holidays+Flickr100k dataset. This dataset uses the same training dataset with Holidays.

Oxford5k: The Oxford5k dataset [72] consists of 5,063 images of buildings and 55 query images corresponding to 11 distinct buildings in Oxford. We follow standard protocol [1], [2]: the bounding boxes of the region of interest are cropped and then used as the queries. As standardly done in the literature, for the learning, we use the Paris6k dataset [74].

The ground truth of queries have been provided with the datasets [71], [72]. Follow the state of the art [22], [37], we evaluate the performance of methods with mAP.

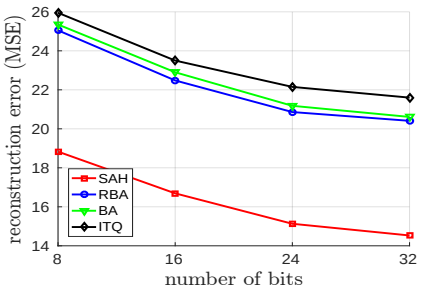


Fig. 4. Reconstruction error (Mean Square Error – MSE) comparison of different methods on Oxford5k dataset

B. Experiments with SIFT features

Follow state-of-the-art image retrieval systems [1], [13], [15], to describe images, we extract SIFT local descriptors [16] on Hessian-affine regions [75]. RootSIFT variant [14] is used in all our experiments. Furthermore, instead of directly using SIFT local features, as a common practice, we enhance their discriminative power by embedding them into high dimensional space (i.e., 1024 dimensions) with the state-of-the-art triangulation embedding [1]. As results, the set of triangulation embedded vectors $\mathcal{V} = \{\mathbf{V}_i\}_{i=1}^m$ is used as the input for the proposed SAH. In order to make a fair comparison to other methods, we aggregate the triangulation embedded vectors with GMP [21] and use the resulted vectors as inputs for compared hashing methods.

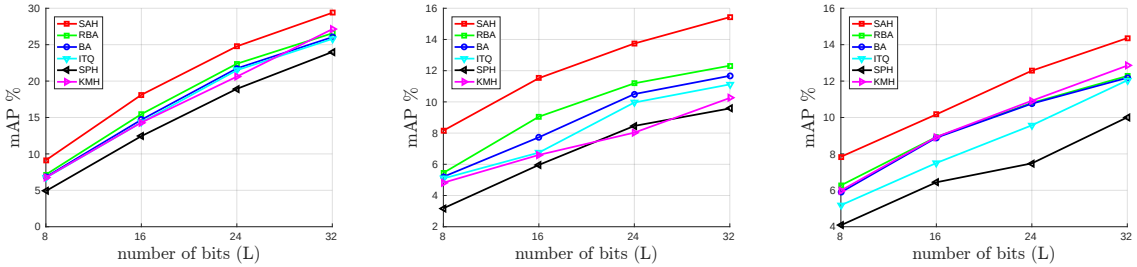
1) Reconstruction comparison: In this experiment, we evaluate the reconstruction capacity of binary codes produced by different methods: ITQ [37], BA [22], RBA, and SAH. We compute the average reconstruction error on the Oxford5k dataset.

For ITQ, BA, and RBA, given the binary codes $\mathbf{Z}$ of the testing data (Oxford5k), the reconstructed testing data is computed by $\mathbf{X}_{res} = \mathbf{W}_2\mathbf{Z} + \mathbf{c}_2\mathbf{1}^T$, where $(\mathbf{W}_2, \mathbf{c}_2)$ is decoder. Note that the decoder is available in the design of BA/RBA and is learned in learning process. For ITQ, there is no decoder in its design, hence we follow [22], i.e., we compute the optimal linear decoder $(\mathbf{W}_2, \mathbf{c}_2)$ using the binary codes of the training data (Paris6k).

For SAH, given the binary codes $\mathbf{Z}$, we use the learned encoder and decoder to compute the aggregated representations Φ by using (21). The reconstruction of Φ is computed by using the decoder as $\Phi_{res} = \mathbf{W}_2\mathbf{Z} + \mathbf{c}_2\mathbf{1}^T$.

Figure 4 shows that BA and RBA are comparable while SAH dominates all other methods in term of reconstruction error. This confirms the benefit of the jointly learning of aggregating and hashing in the proposed SAH.

2) Retrieval results: Figure 5 shows the comparative mAP between compared methods when using SIFT features. We find the following observations are consistent on three datasets. The proposed RBA is competitive or slightly outperforms BA [22], especially on Oxford5k dataset. The proposed SAH improves other methods by a fair margin. The improvement is more clear on Holidays and Oxford5k, e.g., SAH outperforms the most competitor RBA 2%-3% mAP at all code lengths.



(a) Holidays (b) Oxford5k (c) Holidays+Flickr100k

Fig. 5. mAP (%) comparison between SAH and state-of-the-art unsupervised hashing methods when using SIFT features on Holidays, Oxford5k, and Holidays+Flickr100k. GMP [21] is used to aggregate embedded local SIFT vectors to produce global vectors used as inputs for the compared methods.

C. Experiments with CNN feature maps

Recently, in [76]–[78] the authors showed that the activations from the convolutional layers of a convolutional neural network (CNN) can be interpreted as local features describing image regions. Motivated by those works, in this section we perform the experiments in which activations of a convolutional layer from a pre-trained CNN are used as an alternative to SIFT features. It is worth noting that our work is the first one that evaluates hashing on the image representation aggregated from convolutional features.

1) Evaluation protocol: We extract the activations of the 5^{th} convolutional layer (the last convolutional layer) of the pre-trained VGG-16 network [51]. Given an image, the activations form a 3D tensor of $W \times H \times C$, where $C = 512$ which is number of feature maps and $W = H = 37$ which is spatial size of the last convolutional layer. By using this setting, we consider that each image is represented by $37 \times 37 = 1,369$ local feature vectors with dimensionality 512. In [77], the authors showed that the convolutional features are discriminative, hence the embedding step is not needed for these features. Therefore, we directly use the convolutional features as the inputs for the proposed SAH. In order to make a fair comparison between SAH and other hashing methods, we aggregate the convolutional features with GMP [21] and use the resulted vectors as the inputs for compared hashing methods.

2) Retrieval results: Figure 6 shows the comparative mAP between methods. The results show that BA [22], KMH [38] and RBA achieve comparative results. The results also clearly show that the proposed SAH outperforms other methods by a fair margin. The improvements are more clear with longer codes, e.g., SAH outperforms BA [22] 2%-3% mAP at $L = 32$ on three datasets. It is worth noting from Figure 6 and Figure 5 that at low code length, i.e., $L = 8$, SIFT features and convolutional features give comparable results. However, when increasing the code length, the convolutional features significantly improves over the SIFT features, especially on Holidays and Holidays+Flickr100k datasets. For example, for SAH on Holidays+Flickr100k, the convolutional features improves mAP over the SIFT features about 5%, 10%, 14% for $L = 16, 24$ and 32, respectively.

D. Comparison with fully-connected features

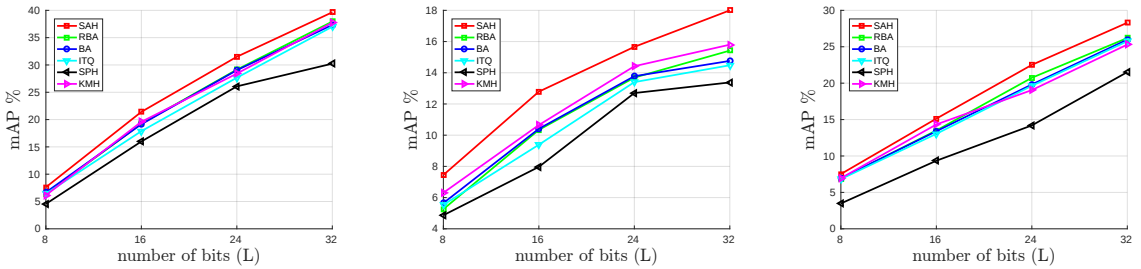
1) Evaluation protocol: In [79], the authors showed that for image retrieval problem, using fully-connected features produced by a CNN outperforms most hand-crafted features such as VLAD [13], Fisher [28]. In this section, we compare the proposed SAH with state-of-the-art unsupervised hashing methods which take the fully-connected features (e.g., outputs of the 7^{th} fully-connected layer from the pre-trained VGG-16 network [51]) as inputs. For the proposed SAH, we take the convolutional features of the 5^{th} convolutional layer of the same pre-trained VGG-16 network as inputs to demonstrate the benefit of the jointly learning of aggregating and hashing.

2) Retrieval results: Figure 7 presents the comparative mAP between methods. At low code length, i.e., $L = 8$, SAH is competitive to other methods. However, when increasing the code length, SAH outperforms compared methods a large margin. The significant improvements are shown on Holidays and Holidays+Flickr100k datasets, e.g., at $L = 32$, the improvements of SAH over BA [22] are 8% and 11.4% on Holidays and Holidays+Flickr100k, respectively.

From Figures 6 and 7, we can observe that for the compared unsupervised hashing methods, using the aggregated local convolutional features instead of fully-connected features can help to achieve significant gains in performance. This indicates the clear advantage of using convolutional features for the retrieval task. And hence, jointly learning to aggregate convolutional features and hashing is very beneficial. Additionally, we summarize the retrieval performance of SAH when using SIFT and convolutional features (Conv.) on different datasets in Table II. Generally, the SAH with convolutional features unsurprisingly achieves better performance than SAH with SIFT. Furthermore, the performance gaps increase as code lengths increase.

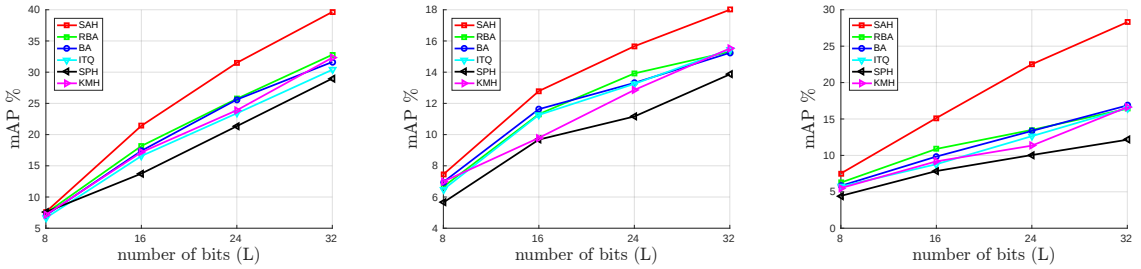
E. Comparison with the recent deep learning-based unsupervised hashing methods

1) DeepBit [59], [80]: In [59], the authors proposed an end-to-end CNN-based unsupervised hashing approach, named DeepBit. To the best of our knowledge, this is the first work using end-to-end CNN for unsupervised hashing. Starting with the pre-trained VGG network [51], the authors replaced the softmax layer of VGG with their binary layer and enforced



(a) Holidays (b) Oxford5k (c) Holidays+Flickr100k

Fig. 6. mAP (%) comparison between SAH and state-of-the-art unsupervised hashing methods when using convolutional features on Holidays, Oxford5k, and Holidays+Flickr100k. Note that GMP [21] is used to aggregate local convolutional feature vectors to produce global vectors as inputs for the compared methods.



(a) Holidays (b) Oxford5k (c) Holidays+Flickr100k

Fig. 7. mAP (%) comparison between SAH and state-of-the-art unsupervised hashing methods using fully-connected features on Holidays, Oxford5k, and Holidays+Flickr100k. Note that SAH still takes the convolutional features as the input.

TABLE II
SAH RETRIEVAL PERFORMANCE (MAP %) WHEN USING SIFT AND CONVOLUTIONAL FEATURES AS INPUTS ON HOLIDAYS, OXFORD5K, AND HOLIDAYS+FLICKR100K DATASETS.

Dataset	Feature	8 bits	16 bits	24 bits	32 bits
Holidays	SIFT	9.13	18.11	24.77	29.38
	Conv.	7.61	21.41	31.50	39.65
Oxford5k	SIFT	8.16	11.52	13.74	15.43
	Conv.	7.45	12.79	15.66	18.00
Holidays +Flickr100k	SIFT	7.84	10.17	12.57	14.36
	Conv.	7.52	15.13	22.51	28.30

several criteria on the binary codes learned at the binary layer, i.e., binary codes should: minimize the quantization loss with the output of the last VGG's fully connected layer, be distributed evenly, be invariant to rotation. Their network is fine-tuned using 50k training samples of CIFAR10. Note that as DeepBit is unsupervised, no label information is used during fine-tuning. Recently, in [80], the authors have improved their DeepBit by using data augmentation to enhance the scale invariance and translation invariance of learned binary codes. The comparative mAP between DeepBit, its improved version (DeepBit-imp.) and other methods on the top $1,000$ returned images (with the class label ground truth) on the testing set of CIFAR10 is cited in the top part of Table III.

2) GraphBit [81]: In [81], the authors proposed to simultaneously learn deep binary descriptors and the structure of a graph, called GraphBit, which represents the interactions (as edges) among different bits. In specific, each bit of the binary descriptor is learned to maximize its mutual information with the input features, while the GraphBit provides additional information where each bit chooses to be instructed by either only inputs or with additional related bits. The retrieval performance of GraphBit in mAP for $1,000$ returned images (as similar to DeepBit) is also presented in the top part of Table III. Note that GraphBit also uses the pretrained VGG [51] as the initial model.

In DeepBit [59] and GraphBit [81], the authors reported results of ITQ, KMH, SPH when the GIST features are used. Here, we also evaluate those three hashing methods on the features extracted from the activations of the last fully connected layer of the same pre-trained VGG [51] with the same setting. These results, i.e., ITQ-CNN, KMH-CNN, SPH-CNN, are presented in the bottom part of Table III. It clearly shows that using fully-connected instead of GIST, ITQ-CNN, KMH-CNN, SPH-CNN have improvements and outperforms DeepBit and GraphBit.

In order to evaluate the proposed SAH, we extract the activations of the last convolutional layer of the same pre-trained VGG and use them as inputs. Similar to DeepBit, we report the mAP on the top $1,000$ returned images. The results of SAH presented in the last row in Table III show that at the same code length, SAH significantly outperforms the recent end-to-end works DeepBit [59], DeepBit improved version [80], and GraphBit [81]. Furthermore, SAH also outperforms ITQ-CNN, KMH-CNN, SPH-CNN with fair margins.

3) Similarity-Adaptive Deep Hashing [60]: Recently, in [60] the authors proposed Similarity-Adaptive Deep Hashing (SADH), an unsupervised hashing method which is based on deep learning. SADH alternatively proceeds over three training modules: deep hash model training, similarity graph

TABLE III
COMPARISON BETWEEN SAH WITH DEEPBIT [59], DEEPBIT IMPROVED VERSION [80], GRAPHBIT [81], AND OTHER UNSUPERVISED HASHING METHODS ON CIFAR10. THE MAP (%) IS COMPUTED ON THE TOP 1000 RETURNED IMAGES.

Method	16 bits	32 bits	64 bits
ITQ [37]	15.67	16.20	16.64
KMH [38]	13.59	13.93	14.46
SPH [39]	13.98	14.58	15.38
DeepBit [59]	19.43	24.86	27.73
DeepBit-imp. [80]	26.36	27.92	34.05
GraphBit [81]	32.15	36.74	39.90
ITQ-CNN	38.52	41.39	44.17
KMH-CNN	36.02	38.18	40.11
SPH-CNN	30.19	35.63	39.23
SAH	**41.75**	**45.56**	**47.36**

TABLE IV
COMPARISON BETWEEN SAH WITH SADH [60] ON CIFAR10. THE MAP (%) IS COMPUTED ON ALL RETURNED IMAGES.

Method	16 bits	32 bits	64 bits
SADH [60]	38.70	38.49	37.68
SAH	36.76	37.85	38.52

updating and binary code optimization. The first module, which is a deep hash model, has a Euclidean loss layer, which measures the discrepancy between the outputs of the deep model and the binary codes learned by the third module. The similarity graph updating module updates the similarity matrix between training images using the current learned deep features of the first module. The binary code optimization module learns binary codes using a graph-based approach, in which the similarity matrix outputted by the second module is used as input. Table IV presents comparative results between the proposed SAH and SADH [60] on the CIFAR10 dataset. The experiment setting is same as SADH [60], in which 100 images per class is randomly sampled and is used as the query set. The rest images are used as the training set. The mAP is computed on all returned images.

The results in Table IV show that SAH achieves competitive results with SADH. Although SADH slightly outperforms SAH at low code length (e.g., 16 bits), at larger code lengths, both methods achieve comparable mAP. However, it is worth noting that performance of SADH is saturated at 16 bits. Its performance is decreased when increasing code length. This fact indicates that SADH potentially suffers from the overfitting problem as more bits are used. Contrary to SADH, the proposed SAH gets improvements when increasing the code length and slightly outperforms SADH at 64 bits. It means that the proposed method is not overfitted over the training set and it is well generalized.

F. Complexity analysis

At testing stage, to compute the binary codes, the proposed SAH includes two steps: (i) compute the aggregating (global) representation φ (Eq. 21) and (ii) encode φ to compute the binary code. In specific, the (asymptotic) complexity for computing (21) is $\mathcal{O}(\max(D^3, D^2 n_i))$, where D is dimension of the local features and n_i is the number of local features in the image. This complexity is similar to the complexity for computing the original Generalized Max Pooling (GMP) (Eq. 2). Additionally, the complexity of the encoding step is $\mathcal{O}(DL)$ which is much smaller than the complexity to compute the aggregated features. We note that given a set of local features, the aggregated features are computed using Eq. (2) before fed to other hashing methods, e.g., ITQ, BA, SPH. That means that the complexity of SAH and other methods are similar when computing the global representation. However, we observe that the running time to compute GMP (Eq. 2) is a bit faster than the one of computing φ (Eq. 21). Table V shows the computational time (ms) of different steps of SAH and other hashing methods. We can observe that SAH takes longer to compute φ than GMP. However, it is still very fast and practical for large scale retrieval systems. In addition, Table V shows that the most expensive step is to extract the convolutional features.

TABLE V
THE COMPUTATIONAL TIME (MS) OF DIFFERENT STEPS OF SAH AND OTHER HASHING METHODS. THE CONVOLUTIONAL FEATURE EXTRACTION IS CONDUCTED WITH A TITAN X GPU USING MATCONVNET [82].

Method	SAH	Other methods
Extract conv. features	≈ 80 ms	
Compute φ	6.3 ms (Eq. 21)	4.1 ms (Eq. 2)
Compute bin. codes from φ	< 1 ms	< 1 ms

Regarding the space requirements, SAH requires to store both encoder and decoder matrices, i.e., $2 \times D \times L$ floating-point numbers where D is the dimension of local features and L is the code length. For local convolutional features (i.e., section V.C), $D = 512$. The $2 \times D \times L$ floating-point space requirement is bigger than the $D \times L$ floating-point space requirement of ITQ, BA. Nevertheless, these space requirements are very small and negligible in comparison with the storage requirement for the pretrained CNN model and the representation of input data.

VI. SIMULTANEOUS FEATURE AGGREGATING AND SUPERVISED HASHING (SASH)

An advantage of the formulation of SAH (18) is the flexibility. When the data label is available, it is possible to extend the unsupervised SAH into its supervised version. In this section, we present the supervised version of SAH, namely simultaneous feature aggregating and supervised hashing (SASH).

A. Formulation

In order to take advantage of the label information, instead of learning binary codes which provide a good reconstruction of aggregated features as in unsupervised version (Section IV), here we aim to learn binary codes which provide a good reconstruction of label vectors. Specifically, we propose to minimize the following constrained objective function

$$\min_{\mathbf{W}_1,\mathbf{c}_1,\mathbf{W}_2,\mathbf{c}_2,\Phi} \frac{1}{2}\left\|\mathbf{Y} - \left(\mathbf{W}_2(\mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T) + \mathbf{c}_2\mathbf{1}^T\right)\right\|^2 + \frac{\beta}{2}\left(\|\mathbf{W}_1\|^2 + \|\mathbf{W}_2\|^2\right) + \frac{\gamma}{2}\sum_{i=1}^{m}\left(\left\|\mathbf{V}_i^T\varphi_i - \mathbf{1}\right\|^2 + \mu\|\varphi_i\|^2\right) \quad (24)$$

$$\text{s.t. } \mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T \in \{-1, 1\}^{L\times m} \quad (25)$$

where $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^m \in \mathbb{R}^{C\times m}$; C is the number of classes; $\mathbf{y}_i \in \mathbb{R}^{C\times 1}$ is the label vector of sample i, in which the index of the maximum element indicates the class of the sample. The proposed constrained objective function (24) has a clear meaning. The term $\mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T$ can be seen as a feature mapping (or encoder) which maps learned aggregated features Φ to binary codes, thanks to the constraint (25). $(\mathbf{W}_2, \mathbf{c}_2)$ can be seen as a linear classifier (or decoder). It takes the codes $(\mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T)$ as inputs and minimizes the l_2 loss w.r.t. the label. The second term of (24) is the regularization on the model weights. Finally, the third term of (24) encourages the aggregating property, i.e., equaling the similarities between the learned aggregated representation φ_i and each column of $\mathbf{V}_i$.

B. Optimization

In order to solve (24) under the constraint (25), similar to SAH (Section IV), we propose to optimize it with alternating optimizing w.r.t. hashing parameters $(\mathbf{W}, \mathbf{c})$ and aggregated representation Φ.

Φ-step: When fixing $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2$ and solving for Φ, we can solve over each sample independently. Specifically, for each sample $i = 1, ..., m$, we solve the following relaxed problem by skipping the binary constraint

$$\min_{\varphi_i} \frac{1}{2}\left\|\mathbf{y}_i - (\mathbf{W}_2(\mathbf{W}_1\varphi_i + \mathbf{c}_1) + \mathbf{c}_2)\right\|^2 + \frac{\gamma}{2}\left(\left\|\mathbf{V}_i^T\varphi_i - \mathbf{1}\right\|^2 + \mu\left\|\varphi_i\right\|^2\right) \quad (26)$$

By solving (26), we find φ_i which not only encourages the aggregating property but also encourages a good classification for a linear classifier, i.e., it minimizes the l_2 loss w.r.t. the label vector. (26) is actually a l_2 regularized least squares problem, hence we achieve the analytic solution as follows

$$\varphi_i = \left((\mathbf{W}_2\mathbf{W}_1)^T(\mathbf{W}_2\mathbf{W}_1) + \gamma\mathbf{V}_i\mathbf{V}_i^T + \gamma\mu\mathbf{I}\right)^{-1} \times \left(\gamma\mathbf{V}_i\mathbf{1} + (\mathbf{W}_2\mathbf{W}_1)^T\left(\mathbf{y}_i - (\mathbf{W}_2\mathbf{c}_1 + \mathbf{c}_2)\right)\right) \quad (27)$$

The asymptotic complexity for computing (27) is $\mathcal{O}(\max(D^3, D^2 n_i))$ which is similar to the asymptotic complexity for computing (2).

$(\mathbf{W}, \mathbf{c})$-step: When fixing Φ and solving for $(\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2)$, (24) under the constraint (25) is equivalent to the following optimization

$$\min_{\{\mathbf{W}_i, \mathbf{c}_i\}_{i=1}^2} \frac{1}{2}\left\|\mathbf{Y} - \left(\mathbf{W}_2(\mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T) + \mathbf{c}_2\mathbf{1}^T\right)\right\|^2 + \frac{\beta}{2}\left(\|\mathbf{W}_1\|^2 + \|\mathbf{W}_2\|^2\right) \quad (28)$$

$$\text{s.t. } \mathbf{W}_1\Phi + \mathbf{c}_1\mathbf{1}^T \in \{-1, 1\}^{L\times m} \quad (29)$$

By solving (28) under the constraint (29), we find hash parameters which not only ensure the binary outputs of the encoded features but also ensure that the learned binary codes give a good classification, i.e., they minimize the l_2 loss w.r.t. the label. (28) and (29) have same forms as (5) and (6) except some changes in variables. Specifically, the first and the second $\mathbf{X}$ in the first term of (5) are replaced by $\mathbf{Y}$ and Φ, respectively; the variable $\mathbf{X}$ in (6) is replaced by Φ. In spite of changing of some variables between these two formulations, the proposed Relaxed Binary Autoencoder (Section III) can be used for solving (28) under the constraint (29). In particular, we reuse the Algorithm 1 for solving $(\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2)$ in which $\mathbf{Y}$ and Φ are used as the inputs for the training. Note that, by changing the parameters, when using RBA (Algorithm 1), the $(\mathbf{W}, \mathbf{c})$**-step** (Section III-B) is changed as follows

$$\mathbf{W}_1 = \lambda\left(\mathbf{B} - \mathbf{c}_1\mathbf{1}^T\right)\Phi^T\left(\lambda\Phi\Phi^T + \beta\mathbf{I}\right)^{-1} \quad (30)$$

$$\mathbf{W}_2 = \left(\mathbf{Y} - \mathbf{c}_2\mathbf{1}^T\right)\mathbf{B}^T\left(\mathbf{B}\mathbf{B}^T + \beta\mathbf{I}\right)^{-1} \quad (31)$$

$$\mathbf{c}_1 = \frac{1}{m}\left(\mathbf{B} - \mathbf{W}_1\Phi\right)\mathbf{1} \quad (32)$$

$$\mathbf{c}_2 = \frac{1}{m}\left(\mathbf{Y} - \mathbf{W}_2\mathbf{B}\right)\mathbf{1} \quad (33)$$

Furthermore, at **B-step** (Section III-B), $\widetilde{\mathbf{X}}$ and $\mathbf{H}$ are computed as follows

$$\widetilde{\mathbf{X}} = \mathbf{Y} - \mathbf{c}_2\mathbf{1}^T \quad (34)$$

$$\mathbf{H} = \mathbf{W}_1\Phi + \mathbf{c}_1 \quad (35)$$

The learning algorithm SASH is similar to the one of SAH and is presented in the Algorithm 3. In the Algorithm 3, $\Phi^{(t)}$, $\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)}$ are values of the corresponding parameters at t^{th} iteration.

Algorithm 3 Simultaneous feature Aggregating and Supervised Hashing (SASH)

Input:
$\mathcal{V} = \{\mathbf{V}_i\}_{i=1}^m$: training data; $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^m$: data label; L: code length; T, T_1: maximum iteration numbers for SASH and RBA (Algorithm 1), respectively; parameters $\lambda, \beta, \gamma, \mu$.

Output:
Hashing parameters $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2$ and aggregated representations before and after learning, i.e., $\Phi^{(0)}$, $\Phi^{(T)}$.

1: Initialize $\Phi^{(0)} = \{\varphi_i\}_{i=1}^m$ with Generalized Max Pooling (2)
2: **for** $t = 1 \to T$ **do**
3: Fix $\Phi^{(t-1)}$, solve $(\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)})$ using Algorithm 1 (which uses $\Phi^{(t-1)}$ and $\mathbf{Y}$ as inputs for training, and the $(\mathbf{W}, \mathbf{c})$**-step** and **B-step** are changed according to (30)-(33), and (34)-(35), respectively.)
4: Fix $(\mathbf{W}_1^{(t)}, \mathbf{c}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_2^{(t)})$, solve $\Phi^{(t)}$ using Φ**-step**.
5: **end for**
6: Return $\mathbf{W}_1^{(T)}, \mathbf{W}_2^{(T)}, \mathbf{c}_1^{(T)}, \mathbf{c}_2^{(T)}, \Phi^{(0)}, \Phi^{(T)}$

C. Mapping from original aggregated features to learned aggregated features and binary codes for new image

1) Mapping from original aggregated features to learned aggregated features: Given set of local features of a test image and learned parameters $(\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2)$, in SASH we can not compute the aggregated representation of the test image using (27) because we do not have the label $\mathbf{y}$ for the test image. To overcome this, we propose a simple yet powerful solution as follows. The Algorithm 3 returns the aggregated representations before and after learning of training images, i.e., $\Phi^{(0)}$ and $\Phi^{(T)}$. For simple notations, we note them as Φ_0 and Φ. We compute (only one time in offline) a linear mapping $\mathbf{P}$ which maps Φ_0 to Φ by solving the following ridge regression

$$\min_{\mathbf{P}} \left(\frac{1}{2} \left\| \Phi - \mathbf{P}\Phi_0 \right\|^2 + \frac{\alpha}{2} \left\| \mathbf{P} \right\|^2 \right) \tag{36}$$

where the regularization term is to prevent overfitting and to obtain a stable solution. We have the analytic solution for $\mathbf{P}$ as follows

$$\mathbf{P} = \Phi\Phi_0^T \left(\Phi_0\Phi_0^T + \alpha\mathbf{I} \right)^{-1} \tag{37}$$

It is worth noting that when learning $\mathbf{P}$, the features Φ have been already available, i.e., Φ has been learned by using the label information (on training images). Hence, we can expect that, the aggregated representations φ of images from the same class will be close together and they are also apart from the aggregated representations of other classes. In the other words, the aggregated representations of samples from the same class will form a compact cluster. That means that $\mathbf{P}$ will map representations φ_0 of images from the same class to the same cluster, i.e., the distance (between the mapped features) of samples from the same class is expected to be small. This is analogous to the distance metric learning [83], [84]. Hence the learning of P can be seen as a simplified distance metric learning which is a well-known approach to the unseen class retrieval [83], [84].

The mapping $\mathbf{P}$ will be used in the process of computing binary codes for new images as the following.

2) Binary codes for new image: Given set of local features of a new image, we first compute its GMP representation φ_0 using (2). We then compute its learned aggregated representation φ by

$$\varphi = \mathbf{P}\varphi_0 \tag{38}$$

where $\mathbf{P}$ is defined by (37). After that, we pass φ to the learned encoder to compute the binary code.

Asymptotic complexity: When computing the binary codes for a new image, it involves three steps, i.e., *(i)* computing φ_0 using (2); *(ii)* computing φ using (38); *(iii)* computing binary codes by the encoder $sgn(\mathbf{W}_1\varphi + \mathbf{c}_1)$. The asymptotic complexities for these three steps are $\mathcal{O}(\max(D^3, D^2 n_i))$, $\mathcal{O}(D^2)$, and $\mathcal{O}(LD)$, respectively. We can see that the most expensive step is to compute φ_0. Hence, the asymptotic complexity when computes φ is similar to the one of GMP (2).

VII. Evaluation of Simultaneous Feature Aggregating and Supervised Hashing (SASH)

In this section, we evaluate and compare the proposed SASH with state-of-the-art supervised hashing methods including Supervised Discrete Hashing (SDH) [44], ITQ-CCA [37], Kernel-based Supervised Hashing (KSH) [42], Binary Reconstructive Embedding (BRE) [41]. For all compared methods, we use the implementations and the suggested parameters provided by the authors. We also compare the proposed SASH to recent end-to-end deep supervised hashing methods, i.e., Deep Quantization Network (DQN) [55], Deep Hashing Network (DHN) [54], Deep Supervised Discrete Hashing (DSDH) [58]. For the proposed SASH, the values of λ (in Eq. (30)), β (in Eq. (30)), and α (in Eq. (37)) are respectively set by cross validation as 10^{-4}, 10^{-3}, and 5×10^{-1} for all experiments. We cross-validate γ and μ (in the objective function (24)) in the ranges of $[10^{-1}, 10]$and $[10^1, 10^5]$, respectively, with the multiplicative step-size of 10.

A. Dataset

Follow the state of the art [42], [44], [53], we evaluate the proposed method on the standard supervised hashing benchmarks CIFAR10 [67], MNIST [68] and NUS-WIDE [85]. The descriptions of CIFAR10, MNIST datasets have been presented in the Section III-C1. The description of NUS-WIDE dataset is provided in the following. In order to configure the training and testing splits, we follow both traditional configuration and a recent proposed configuration [86].

1) Traditional configuration: We follow the traditional configuration in state-of-the-art supervised hashing methods [42], [44].

CIFAR10 for the CIFAR10 dataset, we randomly sample 100 images per class to form 1,000 query images. The remaining 59K images are used as database images. Furthermore, 500 images per class are sampled from the database to form 5K training images.

MNIST for the MNIST dataset, we randomly sample 100 images per class to form 1,000 query image. The remaining images are used as the training images and database images. Note that as KSH and BRE require a full similarity matrix when training, it is difficult for these methods to handle large training data. Follow SDH [44], we sample 5,000 images for training these methods.

NUS-WIDE [85] dataset contains about 270,000 images collected from Flickr. NUS-WIDE is associated with 81 ground truth labels, with each image containing multiple semantic labels. We define the groundtruths of a query as the images sharing at least one label with the query image. As in [42], [44], we select the 21 most frequent labels. For each label, we randomly sample 100 images for the query set. The remaining images are used as database images. Furthermore, 500 images per class are randomly sampled from the database to form the training set.

2) "Retrieval of unseen classes" configuration: Recently, in [86], the authors show that in the traditional configuration, by using same class labels for both training and testing phases, that configuration is more related to the classification task, rather than the retrieval task. Hence, to evalute the proposed method for the retrieval context, which is the focus of this paper, we also follow the "retrieval of unseen classes" configuration that is proposed in [86]. For CIFAR10 and MNIST, we start from their original splits (i.e., the number of training and testing images are 50K and 10K for CIFAR10, 60K and 10K for MNIST) but we use separate classes at training and testing stages. Specifically, 70% of the class labels, which are randomly sampled, are used when learning the hashing function, and the 30% remaining class labels (unseen classes)

TABLE VI
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON CIFAR10 DATASET UNDER TRADITIONAL CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	35.44	38.85	41.21	43.77	45.30
KSH [42]	32.80	37.55	39.13	40.57	41.81
BRE [41]	19.15	22.37	24.11	25.59	26.50
SDH [44]	40.35	43.83	45.92	47.56	48.69
SASH	53.57	57.45	61.33	62.82	63.65

TABLE VII
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON MNIST DATASET UNDER TRADITIONAL CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	58.18	64.24	68.07	69.42	70.92
KSH [42]	65.40	73.81	76.09	78.44	79.52
BRE [41]	27.17	32.27	38.38	40.24	42.37
SDH [44]	69.23	76.31	78.12	80.91	81.63
SASH	75.48	79.34	81.02	83.63	84.51

TABLE VIII
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON NUS-WIDE DATASET UNDER TRADITIONAL CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	56.59	59.98	61.45	61.75	62.45
KSH [42]	62.97	66.51	67.42	67.59	67.94
BRE [41]	34.87	36.34	37.19	38.14	39.94
SDH [44]	63.07	66.89	67.36	67.79	68.10
SASH	64.01	67.10	67.63	68.01	68.75

TABLE IX
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON CIFAR10 DATASET UNDER "RETRIEVAL OF UNSEEN CLASSES" CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	65.48	69.94	70.72	71.33	71.83
KSH [42]	62.41	64.11	62.13	64.40	62.61
BRE [41]	52.55	51.20	53.10	55.73	54.34
SDH [44]	47.43	58.41	61.83	60.08	62.60
SASH	70.08	74.46	77.22	78.33	79.86

TABLE X
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON MNIST DATASET UNDER "RETRIEVAL OF UNSEEN CLASSES" CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	52.40	56.36	58.71	59.82	60.25
KSH [42]	50.27	55.64	57.21	58.31	58.85
BRE [41]	50.03	53.43	55.52	56.91	57.11
SDH [44]	53.64	55.80	57.31	58.14	59.69
SASH	59.10	62.20	63.31	63.94	64.21

TABLE XI
MAP (%) COMPARISON BETWEEN SASH AND STATE-OF-THE-ART SUPERVISED HASHING METHODS ON NUS-WIDE DATASET UNDER "RETRIEVAL OF UNSEEN CLASSES" CONFIGURATION.

L(bits)	8	16	24	32	48
CCA-ITQ [37]	37.81	42.93	43.92	44.31	44.85
KSH [42]	36.18	38.84	39.88	40.08	40.98
BRE [41]	34.10	33.95	35.22	36.14	35.79
SDH [44]	34.92	43.45	43.35	43.89	44.55
SASH	42.62	44.26	45.05	45.43	46.71

are used to evaluate the hashing scheme. We call train70/test70 the train/test images of the 70% classes and train30/test30 the remaining ones. The train70 is used to train the hash function. The test30 is used as queries. The train30 is used as database for the retrieval. In summary, the train70 - train30 - test30 is equivalent to the learn - database - query. The test70 is not used at all.

For NUS-WIDE dataset, we start from the data split of the traditional configuration, i.e. the testing set consists of 100 images per class, the remaining images form database. We randomly split the 21 most frequent labels into 2 groups of 70% (i.e. 15) classes for training and 30% (i.e. 6) classes for unseen retrieval testing. As a result, we have database70/database30 and test70/test30 sets. Note that since each image may have multiple labels, we post-process the test30 and database30 by removing images that contain labels appeared in the set of 70% seen classes. This ensures that there is no overlap in the class labels between training images and testing images. We then sample 500 images for each class from database70 to form train70 set, which is used to train the hash function. The test30 and database30 are respectively used as query set and database for testing.

B. Evaluation protocol

As standardly done in the literature, the retrieval accuracy is reported in term of mean Average Precision (mAP). For all datasets, the labels of images are used as the groundtruth.

Similar to SAH in Section V-C1, we use the 5^{th} convolutional features of the pre-trained VGG network [51] as the inputs for the proposed SASH. In order to make a fair comparison between SASH and other hashing methods, i.e., KSH, BRE, CCA-ITQ, SDH, we aggregate the convolutional features with GMP [21] and use the resulted vectors as the inputs for compared hashing methods.

C. Retrieval results

1) Results on the traditional configuration: Tables VI and VII present comparative results between the proposed SASH and compared supervised hashing methods on the CIFAR10 and MNIST datasets under the traditional configuration. The results show that SASH significantly outperform the compared methods on both datasets. The most competitive method is SDH [44]. The improvement of SASH over SDH is more clear on the CIFAR10 dataset, i.e., from 13%-15% at different code lengths. Table VIII presents comparative results on the NUS-WIDE dataset. The results show that the proposed SASH, SDH [44], and KSH [42] achieve competitive results. Specifically, SASH achieves slightly higher performances than SDH and KSH, and these three methods significantly outperform other methods, e.g., CCA-ITQ, BRE.

2) Results on the "retrieval of unseen classes" configuration: Tables IX and X present comparative results between the proposed SASH and compared supervised hashing methods on the CIFAR10 and MNIST datasets under

TABLE XII
MAP (%) COMPARISON BETWEEN SASH AND END-TO-END DEEP LEARNING-BASED SUPERVISED HASHING METHODS ON CIFAR10, MNIST, AND NUS-WIDE UNDER "RETRIEVAL OF UNSEEN CLASSES" CONFIGURATION.

	CIFAR10			MNIST			NUS-WIDE		
L(bits)	24	32	48	24	32	48	24	32	48
DQN [55]	75.15	76.05	76.78	59.68	61.32	63.76	43.37	43.78	44.24
DHN [54]	74.53	75.95	76.59	60.10	62.57	63.43	44.65	45.03	44.98
DSDH [58]	74.37	75.41	76.23	62.78	63.18	64.05	39.05	40.02	40.58
SASH	77.22	78.33	79.86	63.31	63.94	64.21	45.05	45.53	46.71

the "retrieval of unseen classes" configuration. Under this configuration, the proposed SASH outperforms compared methods a fair margin. The most competitive method is CCA-ITQ [37]. On the CIFAR10 dataset, SASH outperforms CCA-ITQ around 4.5% to 8% at different code lengths. On the MNIST dataset, the improvements of SASH over CCA-ITQ vary around 4% to 6.5% at different code lengths. Table XI presents comparative results on the NUS-WIDE dataset. The proposed SASH also outperforms other methods. However, the improvements are lower than those on CIFAR10 and MNIST datasets, i.e., the improvements of SASH over CCA-ITQ vary around 1% to 5% at different code lengths. The higher improvements are observed at low code lengths, e.g., $L = 8$.

Comparison to end-to-end deep supervised hashing methods: Most traditional end-to-end deep supervised hashing methods [53]–[55], [58] consist of the fine-tuning a deep Convolutional Neural Network which is trained for the classification task. As shown in [86], under the traditional configuration in which the training and testing use the same class labels, one can directly encode the output of the classification layer (e.g. a softmax layer) using only $\lceil log_2 C \rceil$ bits, where C is the number of classes. This simple strategy actually outperforms state-of-the-art supervised hashing methods. However, that configuration is suitable for classification problem, not retrieval problem. Hence, here we evaluate and compare the proposed SASH with end-to-end deep hashing methods using "retrieval of unseen classes" setting [86].

Table XII presents comparative results between the proposed SASH and recent end-to-end supervised deep hashing methods DQN [55], DHN [54], DSDH [58]. It is worth noting that in the original corresponding papers, the authors did not report the performance of these methods under unseen class setting. Hence we use the released implementations and the suggested parameters provided by the authors to conduct experiments. Follow [54], [55], we report the results with $L = 24, 32$ and 48 bits. The experimental results show that on the simple MNIST dataset, SASH achieves competitive results to DQN [55] and DHN [54] at higher code lengths, i.e., $L = 32$ and 48, while it outperforms these two methods at lower code length, i.e., $L = 24$. SASH and DSDH [58] achieve competitive results at all code lengths on the MNIST dataset. On the CIFAR10 dataset, SASH clearly outperforms DQN, DHN, DSDH fair margins at all compared code lengths, i.e., SASH outperforms the second best DQN around 2% to 3% at different code lengths. Regarding the NUS-WIDE dataset, the proposed SASH achieves favorable performances over DQN and DHN across all code lengths, while it significantly outperforms DSDH at all compared code lengths.

VIII. CONCLUSION

In this paper, we first introduce the Relaxed Binary Autoencoder (RBA) hashing method in which we obtain analytic solutions for encoder and decoder during the alternating learning. This leads to the efficient training of RBA. After that, we propose a novel unsupervised hashing approach, i.e., SAH, in which the feature aggregating and hashing are designed simultaneously and optimized jointly. The binary codes are learned such that they not only encourage the aggregating property but also ensure a good reconstruction of the inputs. We further propose a supervised version of SAH, namely, SASH, by leveraging the label information when learning binary codes. The binary codes are learned such that they not only encourage the aggregating property but also optimize for a linear classifier. Extensive experiments on benchmark datasets with different image features and different configurations demonstrate that the proposed methods outperforms the state-of-the-art unsupervised and supervised hashing methods.

ACKNOWLEDGEMENT

This work was supported by both ST Electronics and the National Research Foundation(NRF), Prime Minister's Office, Singapore under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec - SUTD Corporate Laboratory).

REFERENCES

[1] H. Jégou and A. Zisserman, "Triangulation embedding and democratic aggregation for image search," in *CVPR*, 2014.
[2] R. Arandjelovic and A. Zisserman, "All about VLAD," in *CVPR*, 2013.
[3] T.-T. Do, T. Hoang, D. L. Tan, H. Le, T. Nguyen, and N.-M. Cheung, "From selective deep convolutional features to compact binary representations for image retrieval," *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2019.
[4] D. M. Chen, S. S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod, "Residual enhanced visual vector as a compact signature for mobile visual search," *Signal Processing*, pp. 2316–2327, 2013.
[5] T.-T. Do, T. Hoang, D. L. Tan, T. Pham, H. Le, N.-M. Cheung, and I. Reid, "Binary constrained deep hashing network for image retrieval without manual annotation," in *WACV*, 2019.
[6] B. Girod, V. Chandrasekhar, D. M. Chen, N.-M. Cheung, R. Grzeszczuk, Y. A. Reznik, G. Takacs, S. S. Tsai, and R. Vedantham, "Mobile visual search," *IEEE Signal Processing Magazine*, pp. 61–76, 2011.
[7] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 place recognition by view synthesis," *TPAMI*, pp. 257–271, 2018.
[8] R. Arandjelovic, P. Gronát, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: CNN architecture for weakly supervised place recognition," *TPAMI*, pp. 257–271, 2018.

[9] A. Torii, J. Sivic, M. Okutomi, and T. Pajdla, "Visual place recognition with repetitive structures," *TPAMI*, pp. 2346–2359, 2015.

[10] T. Sattler, T. Weyand, B. Leibe, and L. Kobbelt, "Image retrieval for image-based localization revisited," in *BMVC*, 2012.

[11] A. Irschara, C. Zach, J. Frahm, and H. Bischof, "From structure-from-motion point clouds to fast location recognition," in *CVPR*, 2009.

[12] N. Tran, D. L. Tan, A. Doan, T.-T. Do, T. Bui, M. Tan, and N.-M. Cheung, "On-device scalable image-based localization via prioritized cascade search and fast one-many RANSAC," *TIP*, 2019.

[13] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *CVPR*, 2010.

[14] R. Arandjelovic and A. Zisserman, "Three things everyone should know to improve object retrieval," in *CVPR*, 2012.

[15] T.-T. Do and N.-M. Cheung, "Embedding based on function approximation for large scale image search," *TPAMI*, pp. 626–638, 2018.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, pp. 91–110, 2004.

[17] T.-T. Do, Q. Tran, and N.-M. Cheung, "FAemb: a function approximation-based embedding method for image retrieval," in *CVPR*, 2015.

[18] Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik, "Learning binary codes for high-dimensional data using bilinear projections," in *CVPR*, 2013.

[19] S. Kim and S. Choi, "Bilinear random projections for locality-sensitive binary codes," in *CVPR*, 2015.

[20] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing: Binary code embedding with hyperspheres," *TPAMI*, pp. 2304–2316, 2015.

[21] N. Murray and F. Perronnin, "Generalized max pooling," in *CVPR*, 2014.

[22] M. A. Carreira-Perpinan and R. Raziperchikolaei, "Hashing with binary autoencoders," in *CVPR*, 2015.

[23] T.-T. Do, D.-K. L. Tan, T. Pham, and N.-M. Cheung, "Simultaneous feature aggregating and hashing for large-scale image search," in *CVPR*, 2017.

[24] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, 2006.

[25] J. Yang, K. Yu, Y. Gong, and T. S. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *CVPR*, 2009.

[26] Y. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *ICML*, 2010.

[27] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, "Aggregating local images descriptors into compact codes," *TPAMI*, 2012.

[28] F. Perronnin and C. R. Dance, "Fisher kernels on visual vocabularies for image categorization," in *CVPR*, 2007.

[29] J. Wang, W. Liu, S. Kumar, and S. Chang, "Learning to hash for indexing big data - A survey," *Proceedings of the IEEE*, 2016.

[30] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *TPAMI*, 2017.

[31] K. Grauman and R. Fergus, "Learning binary hash codes for large-scale image search," *Machine Learning for Computer Vision*, 2013.

[32] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999.

[33] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *ICCV*, 2009.

[34] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *NIPS*, 2009.

[35] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *TPAMI*, pp. 2143–2157, 2009.

[36] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008.

[37] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011.

[38] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *CVPR*, 2013.

[39] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *CVPR*, 2012.

[40] J. Lu, V. E. Liong, and J. Zhou, "Deep hashing for scalable image search," *TIP*, 2017.

[41] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *NIPS*, 2009.

[42] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *CVPR*, 2012.

[43] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *CVPR*, 2014.

[44] F. Shen, C. Shen, W. Liu, and H. Tao Shen, "Supervised discrete hashing," in *CVPR*, 2015.

[45] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," in *ICML*, 2011.

[46] J. Wang, S. Kumar, and S. Chang, "Semi-supervised hashing for large-scale search," *TPAMI*, pp. 2393–2406, 2012.

[47] F. Shen, Y. Yang, L. Liu, W. Liu, D. Tao, and H. T. Shen, "Asymmetric binary coding for image search," *TMM*, 2017.

[48] X. Zhou, F. Shen, L. Liu, W. Liu, L. Nie, Y. Yang, and H. T. Shen, "Graph convolutional network hashing," *TCYB*, 2018.

[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014.

[52] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *AAAI*, 2014.

[53] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *CVPR*, 2015.

[54] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *AAAI*, 2016.

[55] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen, "Deep quantization network for efficient image retrieval," in *AAAI*, 2016.

[56] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *CVPR*, 2015.

[57] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *TIP*, pp. 4766–4779, 2015.

[58] Q. Li, Z. Sun, R. He, and T. Tan, "Deep supervised discrete hashing," in *NIPS*, 2017.

[59] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *CVPR*, 2016.

[60] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen, "Unsupervised deep hashing with similarity-adaptive and discrete optimization," *TPAMI*, 2018.

[61] H. Zhang, M. Wang, R. Hong, and T. Chua, "Play and rewind: Optimizing binary representations of videos by self-supervised temporal hashing," in *ACM MM*, 2016.

[62] T.-T. Do, A.-D. Doan, and N.-M. Cheung, "Learning to hash with binary deep neural network," in *ECCV*, 2016.

[63] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.

[64] G. Hinton, "Neural networks for machine learning," *Coursera, video lectures*, 2012.

[65] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, pp. 1–10, 1966.

[66] J. Nocedal and S. J. Wright, *Numerical Optimization, Chapter 17*, 2nd ed. World Scientific, 2006.

[67] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[68] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[69] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *TPAMI*, pp. 117–128, 2011.

[70] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *IJCV*, pp. 145–175, 2001.

[71] H. Jégou, M. Douze, and C. Schmid, "Improving bag-of-features for large scale image search," *IJCV*, pp. 316–336, 2010.

[72] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *CVPR*, 2007.

[73] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *ECCV*, 2008.

[74] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *CVPR*, 2008.

[75] K. Mikolajczyk and C. Schmid, "Scale and affine invariant interest point detectors," *IJCV*, pp. 63–86, 2004.

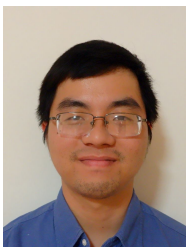
[76] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of CNN activations," in *ICLR*, 2016.

[77] A. Babenko and V. S. Lempitsky, "Aggregating local deep features for image retrieval," in *ICCV*, 2015.

[78] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "From generic to specific deep representations for visual recognition," in *CVPRW*, 2015.
[79] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *CVPRW*, 2014.
[80] K. Lin, J. Lu, C.-S. Chen, J. Zhou, and M.-T. Sun, "Unsupervised deep learning of compact binary descriptors," *TPAMI*, 2018.
[81] Y. Duan, Z. Wang, J. Lu, X. Lin, and J. Zhou, "Graphbit: Bitwise interaction mining via deep reinforcement learning," in *CVPR*, 2018.
[82] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *ACM MM*, 2015.
[83] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell, "Distance metric learning with application to clustering with side-information," in *NIPS*, 2002.
[84] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *ECCV*, 2016.
[85] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "Nus-wide: A real-world web image database from national university of singapore," in *ACM Conference on Image and Video Retrieval*, 2009.
[86] A. Sablayrolles, M. Douze, N. Usunier, and H. Jegou, "How should we evaluate supervised hashing?" in *ICASSP*, 2017.

Thanh-Toan Do is currently a Lecturer at the Department of Computer Science, the University of Liverpool (UoL), United Kingdom. He obtained Ph.D. in Computer Science from INRIA, Rennes, France in 2012. Before joining UoL, he was a Research Fellow at the Singapore University of Technology and Design, Singapore (2013 - 2016) and the University of Adelaide, Australia (2016 - 2018). His research interests include Computer Vision and Machine Learning.

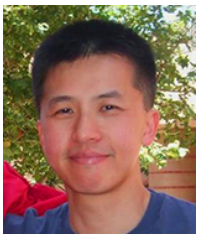
Khoa Le received the BSc for an honours degree from the University of Science, Vietnam National University, in 2015. Since 2016, he has been a research assistant at Singapore University of Technology and Design (SUTD). His current research interests are deep learning and image retrieval.



Tuan Hoang is currently a Ph.D. student at Singapore University of Technology and Design (SUTD) Jan 2016. Before joining SUTD, he achieved the bachelor degree in Electrical Engineering at Portland State University, in 2014. His research interests are content-based image retrieval and image hashing.

Huu Le received the B.S. degree in Electrical and Computer Engineering from Portland State University, Oregon, USA, in 2011 and the Ph.D. degree in Computer Science from the University of Adelaide, Australia, in 2018. He is currently a postdoctoral researcher at Chalmers University of Technology, Sweden. His research interests include robust estimation, non-rigid registration, computational geometry, large-scale image retrieval and optimization methods applied to the fields of computer vision.

Tam V. Nguyen is an Assistant Professor at Department of Computer Science, University of Dayton. He received PhD degree in National University of Singapore (NUS) in 2013. His research topics include computer vision, applied deep learning, multimedia content analysis, and mixed reality. He has authored and co-authored 50+ research papers with 900+ citations according to Google Scholar. His works were published at IJCV, IEEE T-IP, IEEE T-MM, IEEE T-CSVT, Neurocomputing, ECCV, IJCAI, AAAI, and ACM Multimedia. He is an IEEE Senior Member.



Ngai-Man Cheung received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, in 2008. He is currently an Associate Professor with the Singapore University of Technology and Design (SUTD). From 2009 - 2011, he was a postdoctoral researcher with the Image, Video and Multimedia Systems group at Stanford University, Stanford, CA. He has also held research positions with Texas Instruments Research Center Japan, Nokia Research Center, IBM T. J. Watson Research Center, HP Labs Japan, Hong Kong University of Science and Technology (HKUST), and Mitsubishi Electric Research Labs (MERL). His work has resulted in 10 U.S. patents granted with several pending. His research interests include signal, image, and video processing, and computer vision.